Jagetia, M., & Kocak, T. (2004). A novel scrambling algorithm for a robust WEP implementation [wired equivalent privacy protocol]. *IEEE 59th Vehicular Technology Conference, 2004 (VTC 2004-Spring)*, *5*, 2487 - 2491. https://doi.org/10.1109/VETECS.2004.1391370

A Novel Scrambling Algorithm for a Robust WEP Implementation

Mohit Jagetia and Taskin Kocak,
Department of Electrical and Computer Engineering,
University of Central Florida, Orlando, FL 32816-2450.
e-mail: tkocak@cpe.ucf.edu.

Abstract— **Internet enabled wireless devices continue to proliferate and are expected to surpass traditional Internet in the near future. However, data security and privacy remains major concerns in the current generation of wireless connectivity. Wired Equivalent Privacy (WEP) protocol used within the 802.11 standard has "major security flaws" thus WLANs using the protocol are vulnerable to attacks. In this paper, we propose a scrambling algorithm that reduces the vulnerability of the WEP. Both the software and hardware implementations of the algorithm reveal at least 10,000 times improvement in security.**

Keywords: wireless LAN, 802.11, security, WEP.

I. INTRODUCTION

Wireless technology has become an integral part of today's life. The use of wireless networking is rapidly rising with an ever-increasing need for businesses to cut costs and to provide mobility to workers. The wireless technology has spread to devices from small-embedded systems to large general purpose PCs. This is due to cheaper prices, faster speeds and also due to the need for greater mobility. It is desirable to have as much data privacy as possible. Hence in today's networked world security is at a premium. Wireless network is very essential, as it is not bound to any region. Any unauthorized person can read, change or use the private data. Wired equivalent privacy (WEP) is a security protocol, specified in the IEEE Wireless Fidelity (Wi-Fi) standard, 802.11 [1], that is designed to provide a wireless local area network (WLAN) with a level of security and privacy comparable to what is usually expected of a wired LAN. WEP seeks to establish similar protection to that offered by the wired network's physical security measures by encrypting data transmitted over the WLAN. Data encryption protects the vulnerable wireless link between clients and access points; once this measure has been taken, other typical LAN security mechanisms such as password protection, end-to-end encryption, virtual private networks (VPNs), and authentication can be put in place to ensure privacy. However, WEP has "major security flaws" thus WLANs using the protocol are vulnerable to attacks [2,3]. These so called wireless equivalent privacy attacks show themselves in the form of intercepting and modifying the transmissions, and gaining access to restricted networks. In this paper, we propose an algorithm to patch WEP protocol against these attacks.

II. THE PROBLEM STATEMENT AND OUR APPROACH

WEP uses RC4 encryption algorithm [4], which operates by expanding a short key into an infinite pseudo-random key stream. If an attacker flips a bit in the cipher text, then upon decryption, the corresponding bit in the plaintext will be flipped. If an eavesdropper intercepts two cipher text encrypted with the same key stream, it is possible to obtain the XOR of the two plaintexts. Knowledge of this XOR can enable statistical attacks to recover the plaintexts. The statistical attacks become increasingly practical as more cipher text that use the same key stream are known. Once one of the plaintexts becomes known, it is trivial to recover all of the others. To ensure that a packet has not been modified, WEP uses an Integrity Check Value (ICV) field in the packet. To avoid encrypting two cipher text with the same key stream, an initialization vector (IV) is used to augment the shared key and produce a different RC4 key for each packet. The major attacks to WEP are given as follows:

1. Active attack: Modification of the packet by modifying the ICV.
2. Passive attacks:
   a. Integrity violation by analyzing the IV
   b. Table based attack for decrypting every packet that is sent over the wireless link.

In order to avoid these attacks, we propose a novel Scrambling algorithm, which randomize the data from the unauthorized user by adding some standard randomness to it. This random characteristic is a function of the private attribute shared between transmitter and receiver only. In this approach the randomness is achieved by RC4 algorithm and distribution of randomness is provided with different algorithms to increase the complexity of rectifying the encrypted data and optimize utilization of randomness.

A. The Algorithms

In the Scrambling Algorithm (SA), a random octet is inserted in a random position. The random position is obtained by RC4 as a function of the secret key. Currently, the octets contain random information, however, we are in the process of developing ways to utilize these octets for further improvement in security of the packets (e.g., dynamically changing secret keys (TKIP)). Octet insertion as shown in Fig.1 is applied to three different fields in the packet format, namely, ICV, IV and

0-7803-8255-2/04/$20.00 ©2004 IEEE.

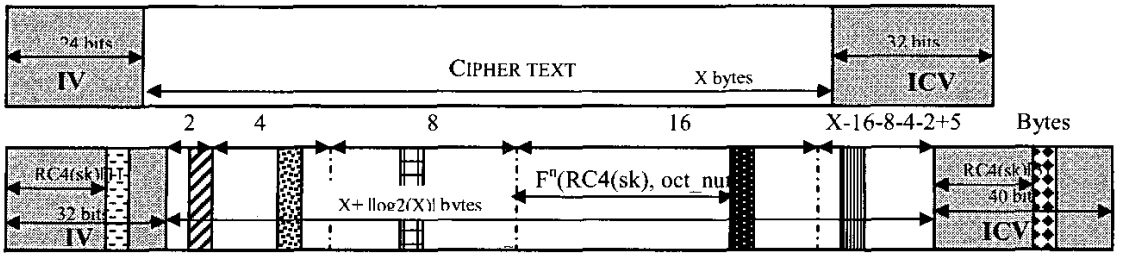


Figure 1: Packet formats for the WEP and modified WEP by the scrambling algorithm

cipher text , to reduce the vulnerability for each of the attacks mentioned above. One octet is inserted for both ICV and IV. However, due to the length and to improve the security, more octets are inserted to the cipher text. In the cipher text, the SA distribute the random octets at random positions in such a way that density of octets reduces along with the length of the packet, which ensures insignificant increase in the packet size for large packets.

Each field (IV, cipher text , ICV) uses one RC4 key-stream octet, to find random position for insertion of random octet. Thus every packet requires 3 RC4 octets from key-stream, so the key stream is divided in set of 3 octets and one set is used for one WEP frame. We have two major algorithms for inserting the random octet at a random position. Algorithm 1, which is used to randomize IV and ICV; is using 5 lower bits of first octet from a set to randomize IV and 5 lower bits of third octet to randomize ICV. Depending upon the value of these 5 bits it inserts random octet in IV and ICV.

Algorithm 1: IV and ICV randomizations / extraction

For entire transmission
Fetch packet_number and IV/ICV from 802.11 protocol
Fetch random data content for the octet from memory
Calculate_random_position(RC4(secret_key)
*[i *packet_number].field_length)*
Insert/Extract the octet at the calculated random position
End for

where *i* is the number of RC4 octets used per packet for randomness.

Algorithm 2 is using the second octet of the set to randomize cipher text. The distributiveAlgorithm used in Algorithm2; uses the LSBs of octet to find the random point in the chunk. Random point is the offset of insertion point in the chunk. The size of the chunk is increased exponentially to utilize the different patterns of the second octet and to create high random density at the starting of the cipher field (explained in section IV). distributiveAlgorithm ensures the insertion of the random octet at the random position throughout the chunk, which is always in the range of 0 to current chunk size. A chunk is a portion of input stream whose size is increasing logarithmically and dependent on the chunk number or chunk position. Each chunk is 2 times in size of its previous chunk and the first chunk is 1 byte wide. This is because every time the random pointer is pre-pended with one bit to point the random position, which results in twice as many points as by the earlier one, so the size is doubled every time. If pre-pending is not binary but rather octal or hex then chunk sizes will be 8 times or 16 times of its predecessor.

Algorithm 2: Cipher text randomization and extraction

For Entire transmission
Fetch packet_number from 802.11 protocol
Reset Cipher_octet_cntr to 0;
Reset octets_processed to 0;
While not end of the cipher text
If (random_position == current_position) then

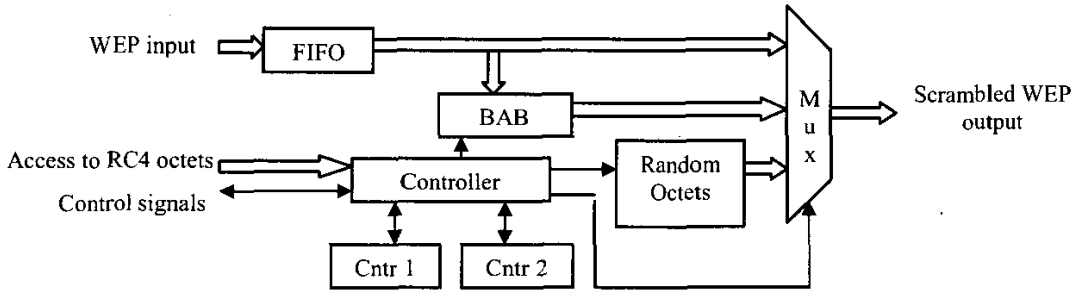


Figure 2: Architecture of the implemented algorithm for scrambling

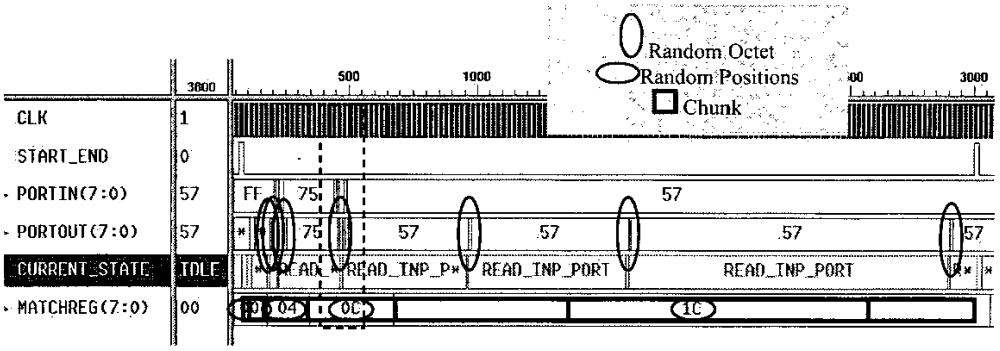


Figure 3: Cipher text randomization

/* (random_position = distributiveAlgorithm (No_of_inserted_octets, packet_number)) == Cipher_octet_cntr */
(Fetch random data content for the octet from memory AND
Insert the octet in the current position) OR (Remove the current octet from stream)
octets_processed = octets_processed+1;
End if
Insert an octet of cipher text ;
Cipher_octet_cntr = Cipher_octet_cntr +1;
Fetch next cipher octet;
End While
End For

SubProc:
Return pos distributiveAlgorithm (No_of_inserted_octets, packet_number)
*pos = RC4(secret_key)[packet_number*i+2][0 to No_of_inserted_octets]*
/* packet_number*i+2 points to the second octet among the current set of RC4 octets used */
End SubProc;

B. Scrambling

Performing the insert action in the above algorithms results in the scrambling of the original WEP cipher text . This action causes the insertion of the random octet at the random position obtained from the distributiveAlgorithm. Insertion could be the part of another protocol enhancing security or other robustness, but must not be the function of Secret Key, because combination of scrambling algorithm and it may cause major leak of secret key information.

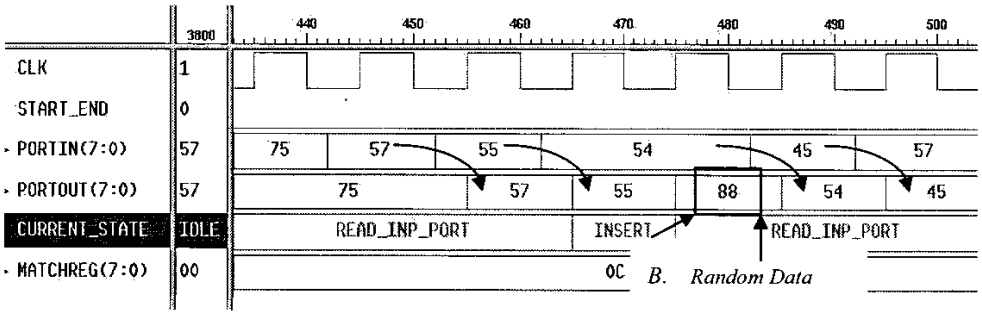
C. DeScrambling

Performing the extract action in the above algorithms results in the Descrambling of the scrambled WEP cipher text. The extracted octet is then made available for other processors if used for other enhancements.

III. Implementation of the algorithMS

The cipher text is randomized in decreasing density of randomness. We are dividing cipher text into virtual chunks of different sizes and adding random contents at random positions in each chunk. The size of a chunk is determined by $2^{(\text{chunk number})}$.

We have implemented the proposed algorithm in MATLAB to verify the functionality at the system level. Furthermore, the algorithm is behaviorally modeled in VHDL to obtain hardware simulation and verification. The architecture of the implemented scrambling algorithm is given in Fig.2. The algorithm works as a post-processor to the WEP protocol. It takes the WEP input and applies randomness to it as specified in section 2. In the architecture, BAB (Bit Addressable Memory Bank) has 2 banks of bit addressable memory. Bank 1 is, of size 32 bits, used to hold input WEP content either IV



B. Random Data

Figure 4: Detailed view of inserting random data in ciphertext randomization

or ICV. Bank 2 is, of size 48 bits, used to hold the resultant randomized output of IV or ICV. Cntr 1 is used to point bits in Bank 1 for the IV starting of the packet and decreases across the length of the packet.

Random insertion positions in above test simulation are calculated by the controller (shown in architecture), which has access to RC4 octets from WEP implementation. In the above test, the RC4 octet has a binary value of "00011100". The last bits of the obtained RC4 octet are used to give different positions in the *m*th chunk between the range of 0 and the maximum chunk size.

The dashed region in Fig. 3 is zoomed in Fig. 4 to illustrate the insertion of random octet in details. In every clock cycle, input is read and forwarded to output if the current state is "read_inp_port" state. When the current state is "insert" state, a control signal is sent to FIFO to wait for a clock cycle; in which the system inserts the random content.

IV. ANALYSIS OF THE SCRAMBLING ALGORITHMS

Notation:

n: number of bytes (octets) received as WEP cipher text
N: number of bytes (octets) result of application of SA
C_S: chunk size
C_P: chunk position
R_O: number of random octets
dR_O: density of random octets

Algorithm 1:

Insertion of an 8-bit random octet in 24-bit IV at any random position, obtained from Calculate_random_position function, results in 6,144 (24*28) different patterns of the same IV. This means an attacker needs to analyze 6,144 more patterns to decrypt the message in case of an IV collision. Thus, the improvement in security is 6,144 times for IV based attacks. The same improvement in the ICV based attacks is 8,192 (32*28) times as ICV is 32 bits long.

Algorithm 2:

Calculation of achieved randomization:

We insert 1 octet per chunk thus

No of octets inserted = No of chunks processed.

No of chunks processed = $1+\log_2$(number of cipher text octets processed)

C_P as a function of incoming octets can be written as

$$C_P(n) = ceil(\log_2(n)) \tag{1}$$

Each insertion has s positions among which one has to be selected randomly where s = chunk size.

Before applying the algorithm, the size of a chunk at position C_P can be calculated as

$$C_S(C_P) = 2^{C_P} \tag{2}$$

Since every chunk has one random octet in it, the total number of random octets inserted can be given as

$$R_O(n) = 1 + C_P(n) \tag{3}$$

For each chunk obtained randomization is:

$$\text{randomization}(C_S) = C_S * 2^8 \tag{4}$$

For C_P chunks the total randomization will be

$$\prod_{i=1}^{C_P} C_S(C_P) \times 2^8 \tag{5}$$

which is the improvement in security for cipher text table based attacks.

The density of inserted random octets in the cipher text can be given as

$$d_{R_O}(n) = \frac{R_O(n)}{n} \tag{6}$$

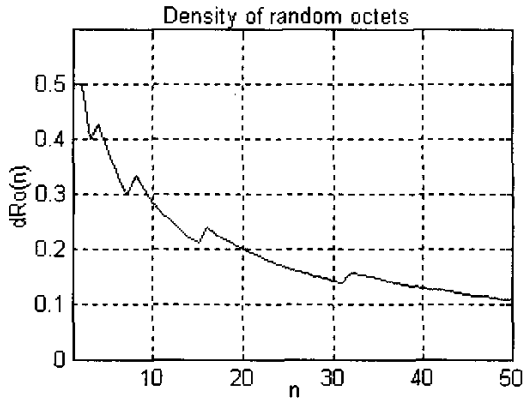


Figure 5: Density of random octets

As depicted in Fig. 5, the density makes a peak at the first 2 bytes of the insertion, and then it reduces logarithmically. The irregular peaks in the curve are caused by the insertion of the random octet. As it requires lesser computational power to retrieve original WEP cipher text from small number of input octets, so we used distributiveAlgorithm to keep high density of randomness at the beginning of frame and reducing logarithmically over the length of the frame.

The number of octets in the modified packet will be given by

$$N(n) = n + R_O(n) \tag{7}$$

Fig. 6 illustrates the change in the cipher part of WEP packet due to the Algorithm2, which shows that for 50 input octets modified cipher will be 56 octets, and for 1024 octets it will be 1035 octets.

Now, let us calculate the probability that an intruder will successfully retrieve the cipher text stream from the scrambled WEP output.

First, the probability of finding a random octet in a chunk can be given as

$$P_{FR}(n) = \frac{1}{C_S(n)+1} \tag{8}$$

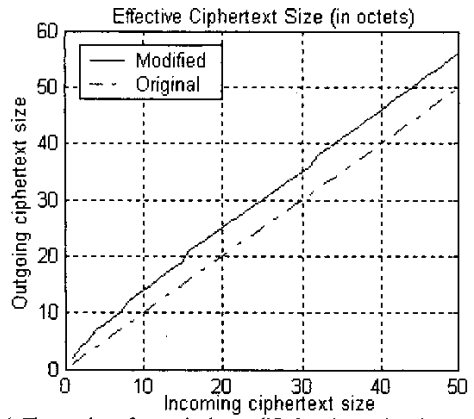


Figure 6: The number of octets in the modified packet against the number of incoming octets.

Then, the probability of successfully retrieving the whole cipher text (i.e., breaking the scrambling algorithm for the cipher text randomization - The probability of finding all random octets) will be

$$P_{SR}(n) = \prod_{i=1}^{n} P_{FR}(i) \qquad (9)$$

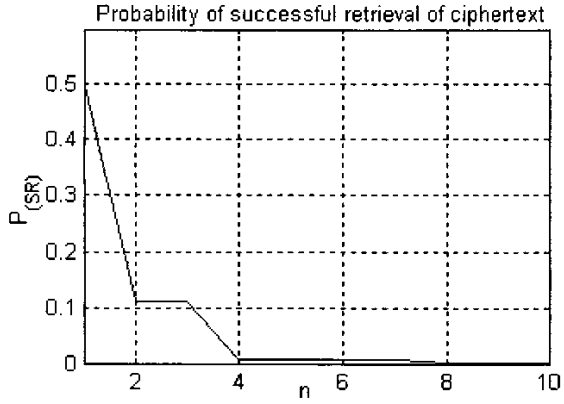


Figure 7: Probability of successful retrieval of cipher text

As the number of cipher text octets increases linearly, the probability of breaking the scrambling algorithm for the cipher text randomization decreases exponentially. For the packets with a cipher text of 5 octets or more, the probability becomes 0.00097 or less.

Finally we calculate the computational difficulty in terms of the number of different patterns generated for same data pattern. Different patterns for the same 24-bit IV can be given as:

$$CD_{IV} = 2^8 \times 24 \qquad (10)$$

Different patterns for the same n data bits can be given as:

$$CD_C(n) = 2^8 \times \log_2(n) \times C_p(n) \qquad (11)$$

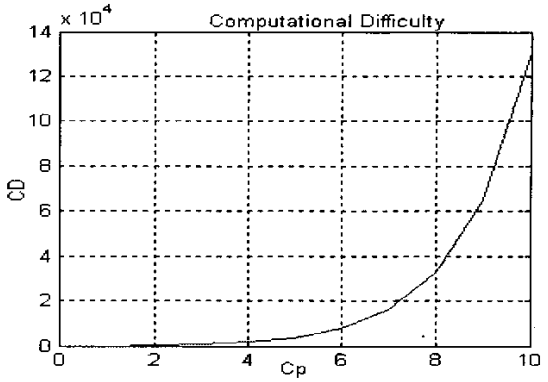


Figure 8: Computational difficulty

Fig. 8 shows the increase in the requirement of computational power to recover the WEP cipher text from the scrambled cipher text.

Different patterns for the same ICV can be given as:

$$CD_{ICV} = 2^8 \times 32 \qquad (12)$$

Increased total computational difficulty can be given as

$$CD\ (n) = CD_{IV} + CD_C(n) + CD_{ICV} \qquad (13)$$

V. Conclusions

A scrambling algorithm is proposed to patch WEP protocol. The algorithm is implemented in MATLAB and VHDL and is verified through simulations. Mathematically, it achieves an aggregate of at least 14848 times (6144 + 8192 + 512(for only one byte cipher text input)) improvement in WEP security. The hardware implementation of the algorithm requires only adding two counters, few registers and a simple controller. Thus the algorithm provides a robust WEP security system without substantially increasing the overall implementation cost.

References

[1] Std 802.11b, IEEE, 1999.

[2] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11", *Proc. of the 7th Annual International Conference on Mobile Computing and Networking*, July 2001.

[3] J. R. Walker, "Unsafe at any key size; an analysis of the WEP encapsulation", *IEEE Document 802.11-00/362*, Oct. 2000.

[4] The RC4 Encryption Algorithm, RSA Data Security, Inc., 1992.